Embedded Applications Journal

A PUBLICATION OF INTEL CORPORATION THIRD QUARTER, 1994

What's Inside

Introducing the 8XC196MH

Christine Neffenger
Applications Engineer
Intel Corporation
Article ID# 0901

We are proud to announce the next proliferation in the motor control family, the 8XC196MH. The motor control family now consists of the 8XC196MC, the 8XC196MD, and the 8XC196MH. These products all operate at 16 MHz at 5 volts. These products are offered in three different packages: 84-lead PLCC, 64-lead SDIP, and 80-lead QFP. These products have a 10-bit analog-to-digital converter, a pulse-width modulator peripheral, and an event processor array for high-speed input capture and output compare. All members of the motor control family have the waveform generator peripheral. The waveform generator generates signals used in AC induction, DC brushless, inverter, and other motor control applications. At the request of the customers, we have added several features to the existing 8XC196MC and 8XC196MD architectures to create the 8XC196MH.

Since motor control algorithms such as vector control are complex and require large amounts of memory for code storage, we doubled the on-chip memory to 32 Kbytes of one-time-programmable memory. For serial communication, the 8XC196MC and 8XC196MD have a peripheral transaction server mode. This mode uses interrupts and the event processor array channels to implement serial communication. The 8XC196MH has two dedicated serial port peripherals, allowing less software overhead. The watchdog timer on the 8XC196MH is the only programmable version of this peripheral in the entire MCS® 96 microcontroller family. The watchdog timer can be programmed with one of four time options. Finally, the waveform generators of the 8XC196MC and 8XC196MD have four modes of operation (two center-aligned PWM modes and two edge-aligned PWM modes). The 8XC196MH has a fifth mode of operation for the waveform generator. This mode enhances the edge-aligned PWM mode by eliminating any jitter present when changing the output.

An overview of the 8XC196MH has been provided here. For an explanation of features and specifications, please order the 8XC196MH data sheet and the 8XC196MH quick reference guide. For additional information on the operation or application examples, order the *8XC196MX User's Manual* and *Application Examples with the 8XC196MC/MD* application note, AP-483, 272282.

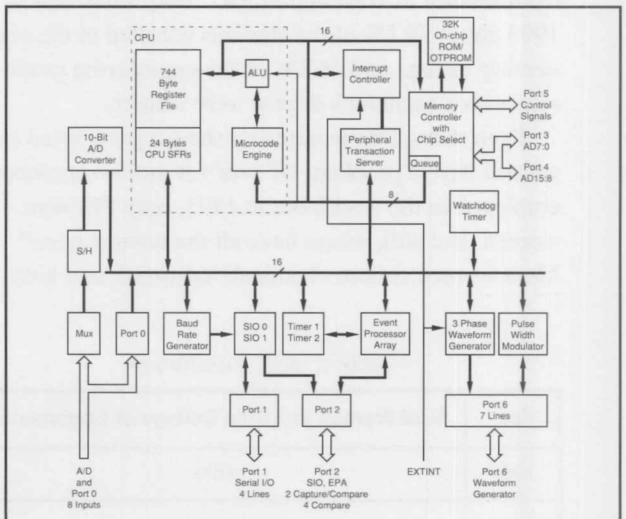
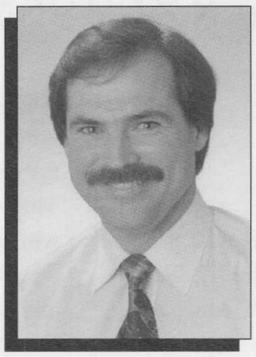


Figure 1. 8XC196MH Block Diagram

From the Managing Editor

Where Have All the Women Gone?



I recently returned from Intel's annual Embedded Systems Seminar series (ESS II), which was held at 18 sites in the U.S. and Canada and 13 sites in Europe. In all of those places, I noticed one startling fact: No women engineers! We had an average of 100 engineers at each site and maybe — maybe — one woman engineer at each site. It doesn't take a genius to figure out that that's less than 1% of the engineering population. I realize this is a small sample of the overall engineering population, but it was still a big surprise to me not to see any women electrical engineers/computer scientists. When I was in school 15 years ago, about 9% of the students in my engineering college were women (the national average, according to the Engineering Workforce Commission's publication, Women in Engineering, May 1994)

I decided to do a little research. I called Arizona State University (ASU) to see whether the engineering enrollment of women has gone up or down. Based on a February 1994 WISE (Women In Sciences and Engineering) newsletter, enrollment for women in all engineering disciplines is up.

I verified this with the Engineering Workforce Commission in Washington, DC. They show that in 1993 about 19.1% of the students enrolled in the engineering college and 16.1% of the engineering graduates with a bachelor's degree were women.

Even though these numbers show a great trend (up), we still have a problem. Of over 1.8 million engineers employed in the workforce in 1993, only 7% were women. But still, where have all the women gone? More women choose chemical, industrial, and civil engineering fields, as compared to electrical and mechanical. Why? Is electrical engineering boring? Is there not enough challenge? Is there not enough money in it? What is it?

I talked to over 20 women in several engineering fields from a variety of companies. Here is what I found out: Some said that electrical engineering is too abstract, that the concepts are not concrete enough. Some said that when a woman chooses an engineering discipline, she sees more women choosing chemical, industrial, or civil and tends to migrate toward those disciplines. Still other women said that mechanical and electrical engineering are male-dominated professions with a long track record of "the good ole boys" and tend not to be conducive to women trying to make a stand. Their inputs are questioned, disregarded, or verified by male members of the team. Whatever the reason, the bottom line is that the whole electrical engineering industry suffers. With only 2,053 undergraduate electrical engineering degrees awarded to women in 1993, making up only 11.7% of the population in electrical engineering, it will be a long time before women can make an impact in the electrical engineering field, from the standpoint of raw numbers.

Facing the facts: We are who we are because of our life experiences and — hold on to your hats, men — "Women are different from men"! They grow up differently, they see the world from a different angle, and

Table 1. ASU Enrollment

Year	% of Women in ASU's College of Engineering
1989	16%
1993	19.4%

*Table 2. Number of undergraduate degrees in 1993**

Discipline	Men	Women	Total	% of Male/Female
Chemical	3150	1524	5674	67.4/32.6
Industrial	2639	1050	3689	71.5/28.5
Civil	7612	1584	9198	82.8/17.2
Computer	3927	748	4675	84.0/16.0
Electrical	15535	2053	17588	88.3/11.7
Mechanical	13448	1661	15109	89.0/11.0

** Engineering Workforce Commission, publication number 133, May 1994, Women in Engineering.*

their parents even have different rules for daughters than for sons. This means that women look at problems and arrive at different solutions because of this different angle. They have formed different opinions. Don't get me wrong — different is good! In this case, different is great! It allows for creative thinking in our work teams. If a team is made up of all one sex, creed, color, background, religion, etc., that team is limiting its creative thought process. Maybe that's why some of the engineering solutions today are simple twists of things already invented, without a lot of radical, earth-shattering inventions. (Just a male, opinionated observation).

*Table 3. Women Engineering Graduates, 1954-1993**

Academic Year	Bachelor's Degrees	Master's Degrees	Doctorates
1954	0.3%	0.3%	0.2%
1964	0.5%	0.3%	0.4%
1974	1.8%	2.5%	1.1%
1984	14.0%	10.1%	4.7%
1993	16.1%	15.7%	9.7%

* *Engineering Workforce Commission, publication number 133, May 1994, Women in Engineering.*

We need to mix up the creative teams shaping the world today. Pull all people together as a team. The more diverse the backgrounds of the individuals, the better. Help get these creative juices flowing! In fact, studies at the University of Western Ontario's School of Business Administration have recently shown that effective use of teams with people from diverse backgrounds can yield a distinct competitive advantage. The researchers found that teams with a high degree of diversity did considerably better than homogeneous teams in an international business setting. They went on to show that more creative solutions to business problems were found and higher-quality decisions were made by members of these teams who learned to communicate effectively across cultural, ethnic, and other boundaries. (The Chronicle of Higher Education, June 1, 1994, Volume XL, Number 39.)

The challenge is ours

Engineering Managers — Create work teams from all walks of life. Work with your local schools to entice more women and minorities into the electrical engineering field. Encourage diverse work groups throughout your companies.

Engineers — Welcome the new blood into the team and make them feel welcome. Listen to what they have to say, no matter how new or "far out" the idea may seem to you. Some of the best ideas start out as "far out."

If we all did these things, who knows what great things could be produced.

Where have all the women gone? Hopefully on your team!

Steven M. McIntyre
Embedded Applications Manager
Embedded Microcomputer Division
Article ID# 0902

More Copies

Need more copies of this issue of the Embedded Applications Journal? Please call Intel Literature Fulfillment at 800-468-8118 in the U.S. and Canada or +44(0)793-431155 in Europe and ask for order #241294-009.

Intel Support Numbers

Customer Support	(U.S. and Canada)	800-628-8686
Customer Training	(U.S. and Canada)	800-234-8806
Literature Fulfillment	(U.S. and Canada)	800-468-8118
Literature Fulfillment	(Europe)	+44(0)793-431155
FaxBack* Service	(U.S. and Canada)	800-628-2283
FaxBack Service	(Europe)	+44(0)793-496646
FaxBack Service	(Worldwide)	916-356-3105
AMO Applications BBS	(Worldwide)	
up to 14.4 Kbaud lines		916-356-3600
dedicated 2400-baud lines		916-356-7209
Applications BBS	(Europe)	+44(0)793-496340

Digital Filter Techniques Using the 80C196KC Eval Board

Cecil A. Moore
Applications Engineer
Intel Corporation
Article ID# 0903

This article is an excerpt from a TechBit that is available from FaxBack (document number 2318). The complete article discusses digital filter theory, provides program listings, and includes references to textbooks and software used in developing the application.

Introduction

Digital signal processing (DSP) is the technique of using digital devices to process continuous signals or data, often in real time. DSP is used in a multitude of applications such as motor control, speech recognition and synthesis, video processing, waveform generation, spectrum analysis, pattern matching, radar/sonar processing, missile guidance, modems, and digital filters. Each generation of higher-speed devices opens new applications for DSP techniques.This article describes a digital filter application designed using a filter design program and an 80C196KC Eval Board.

Using a Filter Design Program

The first step in using a filter design program is to initialize it to the particular processor that is being used. The 80C196 is a 16-bit, fixed-point processor, and that information is entered into the filter design program.The next step is to enter the parameters of the particular filter desired. One possibility is an IIR filter with a sample rate of 1 KHz, a passband of 400-410 Hz, a lower stopband of 200 Hz, and an upper stopband of 490 Hz. We will allow 1 dB of ripple within the passband, and frequencies outside the stopband must have an attentuation of 20 dB. This information is entered into the filter design program and, as if by magic, the filter design software creates everything one needs to implement the filter.

The numbers themselves are normalized to less than one to avoid overflow, but some of the actual values of the coefficients may be greater than one. When this condition occurs, a combination of values less than one and shifts are used where one shift left is the same as multiplying by two and two shifts right is the same as dividing by 4, etc. The input value is also scaled before any operation occurs. The scaling value is the gain of the preamplifier.

80C196 Filter Software

All numbers associated with the digital filter will be signed, hexadecimal fractional numbers with values between -0.9999695 (8000H) and +0.9999695 (7FFFH). 0000H is zero, 0001H is +0.00003051758 (1/32768), and FFFFH is -0.00003051758.

A sine-wave signal generator was connected to JP1-2 (Analog Channel 0) and JP1-1 (ANGND-1) on the 80C196KC Eval Board and adjusted to span 0 volts (min) to 5.12 volts (max). The A/D converter output is 0000H for 0 volts input and FFC0H for 5.12 volts input. Around 2.56 volts, the output changes from 7FC0H to 8000H at half-scale. It would be ideal if the A/D output would go from -1/512 to zero to +1/512 around 2.56 volts, and a simple mathematical trick accomplishes that feat. Simply toggling the sign bit normalizes the A/D output to match the signed hexadecimal fractions that exist in the rest of the filter software.

When a signed hexadecimal (1.15) fractional multiplication is done, the result contains two sign bits (2.30). For instance, multiplying 1/2 (4000H = 214) by 1/2 yields 10000000H = 228, which in signed hexadecimal fractional notation would be 1/8 instead of the required 1/4. To remedy the problem we simply shift left to rid ourselves of one of the sign bits (and multiply our result by two) to obtain the correct fractional value. (2.30 becomes 1.30.)

One nice characteristic of these filters is that the frequency response varies linearly with the sample frequency. The 400-410 Hz bandpass filter becomes a 40-41 Hz bandpass filter if the sample rate is changed from 1000 Hz to 100 Hz. With a sample rate of 20 KHz, the bandpass response is 8–8.2 KHz. Thus a standardized set of filters can be designed and tuned to the required frequency by adjusting the sample frequency. The sampling rate must be greater than twice the signal bandwidth. For our bandpass application, the sampling frequency must be more than twice the upper bandstop frequency. The 1000 Hz sampling frequency is more than twice the upper bandstop frequency of 490 Hz, so we meet the requirements. The phenomenon that occurs when this sampling rule is violated is called aliasing and results in a filter responding to other frequencies as if they were our bandpass center frequency of 405 Hz.

The filters in this article were developed to run on an 80C196KC Eval Board running at 16 MHz. To obtain an analog output that could be viewed on an oscilloscope, an MC1408 D/A converter was mounted on an external breadboard and interfaced to Port 1 through the JP2 connector. Assuming that the A/D converter has interrupted the program and has a digitized value waiting in the A/D Result Register, the resulting 8XC196 interrupt subroutine for the above filter is shown in Figure 1. Of course, assembler directives and initializing code appear before the subroutine.

```
FILTER:
PUSHF
ADD         M00,AD_RESULT,#8000H        ; CHANGE SIGN
LDB         AD_COMMAND,#00H             ; START
HSO,A/D
LDB         HSO_COMMAND,#0FH            ; TIMER1,A/D
ADD         HSO_TIME,TIMER1,#COUNT
MUL         MUL_RESULT,M00,GAIN
SHRAL       MUL_RESULT,#SHG-1
LD          M01,MUL_RESULT+2
MUL         MUL_RESULT,M11,A11
SHLL        MUL_RESULT,#SH1+1
ADD         M01,MUL_RESULT+2
MUL         MUL_RESULT,M21,A21
SHLL        MUL_RESULT,#SH1+1
ADD         M01,MUL_RESULT+2
MUL         MUL_RESULT,M01,B01
SHLL        MUL_RESULT,#SH1+1
LD          M02,MUL_RESULT+2
MUL         MUL_RESULT,M11,B11
SHLL        MUL_RESULT,#SH1+1
ADD         M02,MUL_RESULT+2
MUL         MUL_RESULT,M21,B21
SHLL        MUL_RESULT,#SH1+1
ADD         M02,MUL_RESULT+2
LD          M21,M11
LD          M11,M01
ADDB        PORT1,M02+1,#80H            ; D/A OUT
POPF
```

Figure 1. Interrupt Subroutine

The A/D must be triggered at the sample rate of 1000 Hz. It is very important that the samples from the A/D be equally spaced in time. This can easily be done using the HSO unit and Timer1. At 16 MHz, Timer1 ticks every 1 μs and it takes the A/D converter 19.56 μs to complete a conversion. 19.56 μs is 19.56 ticks. To get a 1000 Hz sample rate, we must start an A/D conversion every 1000 μs, which is 1000 ticks. We load Timer1 with 1000 minus 27 (A/D conversion time plus interrupt latency time) or 973. Note that using the HSO hardware to trigger the A/D function gives equally spaced sample times, while triggering the A/D from a software loop or under interrupt control yields sample time that are less equally spaced. After setting up the HSO and A/D, we simply go into a loop waiting for an interrupt.

```
LD      M11,M01
HERE:
SJMP HERE                   ; wait for interrupt
```

Figure 2. Loop

Obviously, when a digital filter is implemented in a system, the software will be performing other tasks until it is interrupted by the A/D. If the D/A converter is used to generate an output waveform, interrupts should be disabled during the digital filter interrupt routine so that a constant input and output sample time is maintained.

Throughput With Digital Filters

A single two-pole filter running under interrupt control with a sample rate of 1000 Hz consumes 3% of the 16 MHz 8XC196KC's processing power. Eight of these filters running concurrently (one for each A/D input) would consume 23% of the time. A single two-pole filter running with a sample rate of 40 KHz would completely consume the processing power of a 16 MHz 8XC196. The time it takes to do an A/D conversion is almost exactly the time it takes to execute a two-pole filter routine, so the next digitized value is available by the time that we need it.

The fixed overhead is approximately 9 μs, and each set of two poles takes an additional 20 μs when running under interrupt control. Running a tight loop as fast as possible (no interrupts, no polling), the maximum sample rate for a two-pole (20 dB) filter is 40 KHz, for a four-pole (40 db) filter it is 21 KHz, and for a six-pole (80 dB) filter it is 15 KHz. Running under interrupt control at the 1000 Hz sample rate of the filters in this article, a 20 dB filter takes 1/35 of the processor's time, a 40 dB filter takes 1/20 and an 80 dB filter takes 1/15. From the Nyquist Theorum, we can say that the upper limit stopband frequency would be 17 KHz for a two-pole filter, 10 KHz for a four-pole filter, and 7 KHz for a six-pole filter running as fast as possible.

Continued on page 8

Auto-programming the 87C196MC/MD

Jennie Abella
Applications Engineer
Intel Corporation
Article ID# 0904

Introduction

The auto-programming mode is a low-cost programming alternative. Using this mode, the contents of an external EPROM are copied to the internal OTPROM of the 87C196MC/MD. This article explains how to construct an inexpensive programmer for the 87C196MC/MD. A programmer for the 27128 is still necessary, but these are readily available.

Auto Programming Algorithm

The 87C196MC/MD enters a programming mode when V_{PP} (typically 12.5V) is applied to EA# during the rising edge of RESET#. The combination of this rising edge of EA# and the value on the PMODE pins causes the 87C196MC/MD to enter a particular programming mode. Then the following sequence is initiated:

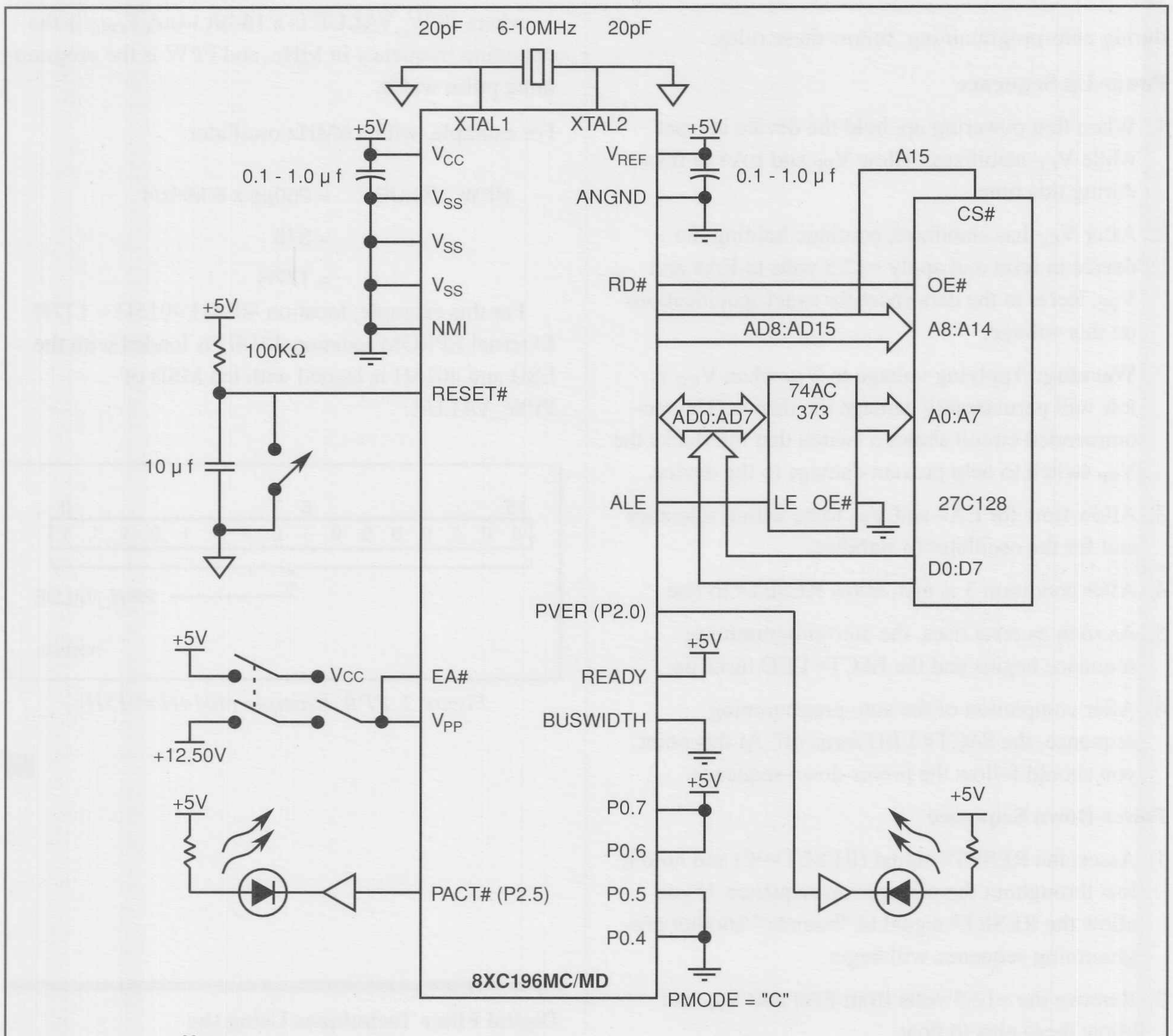
1. The upper port 0 pins P0.4–7 (PMODE pins) are read to determine which programming mode is selected. If the value is 1100 (binary), the auto programming mode is entered.
2. The internal OTPROM location 2018H (CCR security lock bits) is checked to see if the security key has been programmed. If they are programmed (=0), then the internal security key locations (2020H–202FH) are checked with external locations 4020H–402FH. If security is passed, the program continues. If not, the device enters an endless loop and must be reset to exit it. The LED to indicate that programming has started will not light. (Follow the powerdown procedure and then another attempt can be made to program the device.)
3. The PPW is then loaded from external locations 4014H/4015H, which must set up a 250 s programming pulse. (The equation for this PPW value is shown in "Calculating the Programming Pulse Width" in this article.)
4. PACT# is activated (P2.5=low) to indicate that programming has started. (LED will light.)
5. PVER is initialized (P2.0=high) to indicate that there are no errors to this point. (LED will not be lit.)
6. External data is then read, starting at 4000H.
7. If the word is not 0FFFFH, then the modified quick-pulse subroutine is called (Step 8). If the word is 0FFFFH, the word is not programmed and the address is checked to see if programming is completed (Step 10).
8. The PPW timer is started and the data word is written to the OTPROM. The program waits until an interrupt is caused by the PPW timer, which ends the programming pulse. If two writes have not been written, this step is repeated. When two writes have been written to this location, then programming continues with the next word.
9. This location is then read back from the OTPROM and compared to the external location. If it did not program correctly, then PVER is deasserted (P2.0=low) and the LED lights, indicating an error. Programming will continue even if an error is detected. The part cannot be programmed again.
10. The address is checked to see if programming has been completed. If programming is not completed, the address pointer is incremented and the next word is programmed. (This goes back to Step 6 until programming is completed.) If the address is 07FFEH, programming is completed, PACT is deactivated (P2.5=high), and an infinite loop is entered. Continue with the power-down procedure.

Auto-Programming Circuit

Figure 1 is the recommended circuit for auto-programming the 87C196MC/MD. Since BUSWIDTH is low, an 8-bit external bus is used. Code to be programmed in the 87C196MC/MD at 2000H must be

Table 1. Auto-programming Memory Map

External EPROM Address	Internal OTPROM Address	Description
4014H	N/A	PPW least-significant bit
4015H	N/A	PPW most-significant bit
4000H–7FFFH	2000H–5FFFH	Reserved locations for code and data
4020H–402FH	2020H–202FH	Security key, during verification



Notes:
Tie all other inputs to either V_{CC} or V_{SS}.
Allow RESET to rise after V_{CC}, EA#, and V_{PP} are stable. See "Power-up Sequence" for details.

A2566-01

Figure 1. Recommended Auto-programming Circuit

located in the external EPROM starting at location 4000H. The memory remapping is shown in Table 1. The PPW must be located in the external EPROM at location 4014H and 4015H. (See "Calculating the Programming Pulse Width" in this article.)

P0.4–P0.7 are hardwired to V_{SS} and V_{CC} and determine the programming mode. The status outputs PACT# and PVER are buffered by a 74HC14 and drive LEDs to indicate programming active (PACT#) and programming verification (PVER). All unused inputs are connected to ground (V_{SS}) and unused outputs are left floating. READY, NMI, and BUSWIDTH are active and should be connected as indicated.

Auto-programming is specified for a crystal frequency of 6 to 10 MHz. A 27(C)128 EPROM with tACC = 250 ns and tOE = 100 ns or faster specifications should be used.

Power-Up and Power-Down Sequencing

Important: To avoid damaging the 87C196MC/MD

Continued on page 8

Continued from page 7

during auto-programming, follow these rules:

Power-Up Sequence

1. When first powering up, hold the device in reset while V_{CC} stabilizes. Allow V_{PP} and EA# to float during this time.
2. After V_{CC} has stabilized, continue holding the device in reset and apply +12.5 volts to EA# and V_{PP}. Refer to the data sheet for exact specifications on this voltage.

   Warning: Applying voltage to V_{PP} when V_{CC} is low will permanently damage the device. The recommended circuit shows a switch that interlocks the V_{PP} switch to help prevent damage to the device.
3. Allow time for EA# and V_{PP} to be within tolerance and for the oscillator to stabilize.
4. After condition 3 is met, allow RESET# to rise.
5. As soon as reset rises, the auto-programming sequence begins and the PACT# LED turns on.
6. After completion of the auto programming sequence, the PACT# LED turns off. At this point, you should follow the power-down sequence.

Power-Down Sequence

1. Assert the RESET# signal (RESET#=0) and hold it low throughout the powerdown sequence. If you allow the RESET# signal to "bounce," another programming sequence will begin.
2. Remove the +12.5 volts from EA# and V_{PP} and allow these pins to float.

   Warning: You must allow EA# and V_{PP} to float before removing V_{CC}, or the device will be damaged.
3. Turn off the V_{CC} supply and allow time for it to reach 0 volts.
4. You can now remove the device from the auto-programming circuit.

Calculating the Programming Pulse Width

The programming pulse width must be 250s for the device to program correctly. Use the following formula to calculate the PPW_VALUE. Round the PPW_VALUE to the next higher integer value and load this value into the PPW location in the external EPROM.

$$\text{PPW\_VALUE} = \text{PPW} \times (F_{OSC}/4)$$

where PPW_VALUE is a 16-bit word, F_{OSC} is the operating frequency in MHz, and PPW is the programming pulse width.

For example, with a 6MHz oscillator:

$$\begin{aligned}\text{PPW\_VALUE} &= 250\mu s \times 6\ \text{MHz}/4\\ &= 375\\ &= 177\text{H}\end{aligned}$$

For this example, location 4014H/4015H = 177H. External EPROM location 4014H is loaded with the LSB and 4015H is loaded with the MSB of PPW_VALUE.

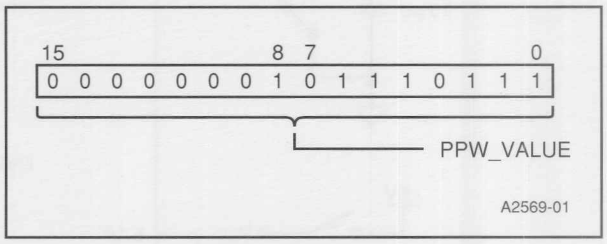


Figure 2. PPW Example (4014H/4015H)

Digital Filter Techniques Using the 80C196KC Eval Board

Continued from page 5

A D/A converter was attached to the 80C196KC Eval Board to observe the output on an oscilloscope. This is only one possibility of what to do with the output. Another approach would be to keep a running average of the absolute value of the output (DFT or FFT). Setting an appropriate threshold would allow for simple detection of the existence of the desired frequency. The filter software can then route the output to either the D/A converter or the Port 1 LEDs.

Summary

This article describes one digital filter application using an 8XC196 microcontroller. For more detailed information and program listings, please consult FaxBack document #2318.

Implementing Additional External Interrupts on MCS® 51 Microcontrollers - Part 1

SS Dang/YY Soh
Applications Engineers
Intel Corporation
Article ID# 0905

Introduction

In most of the Intel MCS® 51 microcontrollers, only two dedicated external interrupt sources (INT0 and INT1) are available for designers. What if the system design requires more than two external interrupt sources? If other interrupt sources such as the PCA or timer 0, 1, and 2 are not being used, they can be configured to serve as additional external interrupt sources. This article describes how to configure the timer 2 interrupt source for this purpose. Upcoming articles will describe the other options.

Timer 2

Timer 2 can serve as an additional external interrupt source by using one of four available modes: capture, 16-bit auto-reload (counter overflow), clock generator, or baud-rate generator. The T2MOD and T2CON registers, shown in Figures 1 and 2, configure

7							0
R	R	R	R	R	R	T2OE	DCEN

A2571-01

Figure 1. T2MOD Register

7							0
TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2#	CP/RL2#

A2570-01

Figure 2. T2CON Register

the timer for these modes.

Capture Mode

When timer 2 is configured for capture mode, a falling edge of an external signal applied to the T2EX pin generates an interrupt that is vectored to address 02BH (the timer 2 interrupt vector). To configure timer 2 for capture mode, program the T2MOD and T2CON registers as follows:

In the T2MOD register, clear the T2OE and DCEN bits (T2OE, DCEN = 0).

In the T2CON register,

- select capture mode (CP/RL2# = 1),
- select timer operation (C/T2# = 0)
- clear the timer run bit (TR2 = 0),
- enable external trigger pin T2EX (EXEN2 = 1),
- clear external capture flag (EXF2 = 0) in both initialization and interrupt service routine, and
- clear unused TCLK, RCLK bits and TF2 flag (TCLK, RCLK, TF2 = 0).

The sample program is shown in Figure 3.

```
$INCLUDE (C51FX.REG)            ;Registers
                                ;declaration
;
            ORG     00H
            AJMP    T2_INIT     ;Timer 2
                                ;initialization
            ORG     02BH
            AJMP    T2_ISR      ;Timer 2 interrupt
                                ;vector
;
;This is Timer 2 initialization routine
;
            ORG     0100H
T2_INIT:    MOV     T2MOD, #00H ;Clear unused bits
            MOV     IP, #20H    ;Set Timer 2
                                ;interrupt priority
            ORL     IE, #0A0H   ;Enable Timer 2 &
                                ;global interrupt
            MOV     T2CON, #09H ;Set Timer 2 in
                                ;capture mode
            SJMP    $           ;Wait for interrupt
;
; This is Timer 2 interrupt service routine
;
T2_ISR:     CLR     T2CON.6     ;Clear interrupt flag,
                                ;EXF2
;           User's routine here
            RETI
            END
```

Figure 3. Sample Program Using Timer 2 in Capture Mode as an Additional External Interrupt Source

The constraints of this implementation are that other timer 2 features cannot be used while it is in this mode and P1.1/T2EX must be dedicated to the external interrupt (as P3.2/INT0# and P3.3/INT1# are for

Continued on page 10

Continued from page 9

external interrupts INT0 and INT1). The external signal is sampled once each machine cycle, so the external interrupt signal applied to T2EX should be held high for at least one machine cycle and then held low for at least one machine cycle to ensure detection of a valid falling edge.

16-bit Auto-reload (Counter Overflow) Mode

When timer 2 is configured for auto-reload (counter overflow) mode, a falling edge of an external signal applied to the T2 pin causes the counter (which was preloaded with FFFFH) to overflow. This counter overflow generates an interrupt that is vectored to address 02BH (the timer 2 interrupt vector). To configure timer 2 for this mode, preload the counter registers (RCAP2H, RCAP2L, TH2, and TL2) with 0FFH, and program the T2MOD and T2CON registers as follows:

In the T2MOD register, clear the T2OE and DCEN bits (T2OE, DCEN = 0).

In the T2CON register,

- select 16-bit auto-reload mode (CP/RL2# = 0),
- select external event counter operation (C/T2# = 1),
- start the counter (TR2 = 1),
- ignore T2EX events (EXEN2 = 0),
- clear unused TCLK, RCLK bits and EXF2 flag (TCLK, RCLK, EXF2 = 0), and
- clear counter overflow flag (TF2 = 0) in both initialization and interrupt routine.

The sample program is shown in Figure 4.

The constraints of this implementation are that no other Timer 2 features can be used while it is in this mode, and P1.0/T2 must be dedicated to the external interrupt. The external interrupt signal applied to T2 should be held high for at least one machine cycle and then held low for at least one machine cycle to ensure detection of a valid falling edge.

Clock Generator or Baud-rate Generator Mode

When timer 2 is configured as a clock generator or a baud-rate generator, a timer rollover does not generate interrupt TF2. With the EXEN2 bit set, a falling edge on pin T2EX generates interrupt EXF2.

```
$INCLUDE (C51FX.REG)                    ;Registers
                                        ;declaration
;
          ORG      00H
          AJMP     T2_INIT              ;Timer 2
                                        ;initialization
          ORG      02BH
          AJMP     T2_ISR               ;Timer 2
                                        ;interrupt vector
;
;This is Timer 2 initialization routine
;
          ORG      0100H
T2_INIT:  MOV      T2MOD, #00H          ;Clear unused bits
          MOV      TH2, #0FFH           ;Preload Timer 2
                                        ;counter registers
          MOV      TL2, #0FFH
          MOV      RCAP2H, #0FFH        ;Timer 2 counter
                                        ;reload value
          MOV      RCAP2L, #0FFH
          MOV      IP, #20H             ;Set Timer 2
                                        ;interrupt priority
          ORL      IE, #0A0H            ;Enable Timer 2 &
                                        ;global interrupt
          MOV      T2CON, #06H          ;Set Timer 2 event
                                        ;counter, run bit
          SJMP     $                    ;Wait for
                                        ;interrupt
;
; This is Timer 2 interrupt service routine
;
T2_ISR:   CLR      T2CON.7              ;Clear interrupt
                                        ;flag, TF2
;         User's routine here
          RETI
          END
```

Figure 4. Sample program Using Timer 2 in 16-bit Auto-reload (Counter Overflow) Mode as an Additional External Interrupt Source

Therefore, it is possible to use timer 2 simultaneously as an external interrupt source and as a clock generator or baud-rate generator. The constraint of either of these implementations is that P1.1/T2EX must be dedicated to the external interrupt. The external interrupt signal applied to T2EX should be held high for at least one machine cycle and then held low for at least one machine cycle to ensure detection of a valid falling edge.

Note that timer 2 can be used as an additional external interrupt source in only one of the four modes described above. The interrupt response time is always greater than 3 machine cycles and less than 9 machine cycles in a single-interrupt system.

Conclusion

The above examples show that timer 2 can be used as an external interrupt source in addition to the two existing, dedicated external interrupt sources.

How to Figure Number of Wait States

Christine Neffenger
Applications Engineer
Intel Corporation
Article ID# 0906

You have a 120 ns memory device and a 20 MHz 8XC196KC device. The 8XC196KC's bus cycles are too fast to interface to the slower memory device. The insertion of wait states into the 8XC196KC bus cycle allows the interface to occur properly. So how do you choose the correct number of wait states?

The following memory timings must be taken into consideration when calculating the number of wait states: the address access time (tACC), the output enable delay (tOE), and the write pulse width (tWP). The names of these specifications can vary from device to device, but the main concept is the same. These memory specifications correspond to the bus cycle specifications of the MCS® 96 microcontroller. The respective specifications on the MCS 96 microcontroller are: the address valid to data valid time (T_{AVDV}), the read low to data valid time (T_{RLDV}), and the write low to write high time (TWLWH).

Let's look at an example. We will use the Intel boot-block Flash 28F001BX-120 and the Intel 8XC196KC 20 MHz device. Table 1 shows the key specifications of these devices.

The 8XC196KC's T_{AVDV} value (106 ns) is too low for the Flash's tACC time (120 ns). We need at least 14 ns more. The Flash's tOE (50 ns) requires 12 ns more from the 8XC196KC's T_{RLDV} (28 ns) spec. The data float delay (tDF) is fine for the system. The 8XC196KC's write pulse width is too small by 20 ns for the Flash. Therefore, we must insert wait states, but how many?

Table 1. Intel Boot Flash 28F001BX-120 and 8XC196KC (20 MHz) Specification Values.

28F001BX-120		8XC196KC (20 MHz)	
	Should be ?	$T_{OSC} = 1/F_{OSC}$ = 50 ns, Latch Delay = 11 ns	
tACC = 120 ns	<	T_{AVDV} = 3TOSC - 55	T_{AVDV} = 95 + latch delay = 106 ns
tOE = 50 ns	<	T_{RLDV} = TOSC - 22	T_{RLDV} = 28 ns
tDF =	<	T_{RHDZ} = TOSC	T_{RHDZ} = 50 ns
tWP =	<	T_{WLWH} = TOSC - 20	T_{WLWH} = 30 ns

The T_{AVDV}, T_{RLDV}, and T_{WLWH} specifications are increased by $2*T_{OSC}*n$, where n is the number of wait states. The numbers given in Table 1 are assuming 0 wait states. One wait state will insert 2*50 ns*1 = 100 ns, two wait states will insert 2*50 ns*2 = 200 ns, etc. Since the largest time we missed the Flash's specification by was 20 ns (T_{WLWH}), we just need to insert 1 wait state or 100 ns into the bus cycle. Table 2 shows the timings with one wait state added.

Table 2. Intel Boot Flash 28F001BX-120 and 8XC196KC (20 MHz) with One Wait State.

28F001BX-120		8XC196KC (20 MHz)
		$T_{OSC} = 1/F_{OSC}$ = 50 ns, Latch Delay = 11 ns
tACC = 120 ns	<	T_{AVDV} + latch delay + wait state = 206 ns
tOE = 50 ns	<	T_{RLDV} + wait state = 128 ns
tDF = 30 ns	<	T_{RHDZ} = 50 ns
tWP = 50 ns	<	T_{WHWL} + wait state = 130 ns

As you can see, we now comfortably meet the Flash memory specifications.

Summary

Calculating wait states is a matter of comparing the key specifications of the memory to the MCS 96 microcontroller's specifications. If the MCS 96 controller is too fast, you can just lengthen the bus cycle by inserting wait states. Remember, though, that you take a performance hit when you insert wait states, but you can interface slower, less expensive memories. ■

Understanding the Development Cycle

By Steven M. McIntyre
Applications Manager
Intel Corporation
Article ID# 0907

The embedded control development cycle is divided into four phases: selection, system evaluation, full development, and production. Each phase within the development cycle requires tools.

Each tool is unique and is designed to fit the needs within that phase of development. Many tools cross over phases, like logic analyzers and programmers, but one thing is for sure: Without tools the task is just a vision.

Disclaimer: This article is the viewpoint of the author and may not reflect the viewpoint of Intel or Intel's third party tools vendors.

1. Selection Phase: Picking the right device for the job.

The Cycle Starts Here

Like any development cycle, it starts with selecting a product to fit the need. Many semiconductor vendors rely on the user to figure

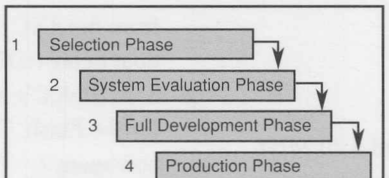


Figure 1. Development Cycle Flow Chart

out the right chip for each application. Others want to make that decision for you. As with any big decision, it is an emotional one. Engineers are making decisions that could impact their companies' bottom line. If they pick a slow device, they may not find out until it is too late in the design cycle to change. If they pick one that is too fast, they could be paying for unnecessary horsepower.

Engineers require tools during the selection phase of development just as much as they need them during the three other phases. Historically, selection tools have been limited. Engineers use product line cards, handbooks, or field technical sales engineers to assist in the selection.

Line cards and handbooks leave much to the engineer. Engineers know their applications, but not every microcontroller made by semiconductor companies. Line cards help narrow the search if the engineers know what they are looking for. The line cards put together by semiconductor vendors are intended to be used by engineers who know what to do with common features found on these devices.

For example, say the application requires information to be shared across a serial link. One would look on a line card for "serial port," right? Line cards limit an engineer's creativity. General-purpose microcontrollers and microprocessors have peripherals that can be used for many things. A simple I/O pin can be used to receive/transmit information in a "serial" manner. It's just a different way to look at it. Simple I/O may be slower, but it should be cheaper than a serial port, wouldn't you think?

Using a semiconductor field applications engineer to make the decision is a little like having a used car salesperson drive a car around the block and tell you that it drove OK, so buy it.

Engineers should get to know their field applications engineers (FAEs). FAEs can help point in the right direction and serve as consultants when times get tough.

As semiconductor vendors increase their portfolios, the search task for design engineers gets tougher. "Which chip is best for my application?" "What peripherals will I need?" "Does the device have enough perfor-

Tools Required for Selection Phase

- ❖ Product line card
- ❖ Handbooks
- ❖ Device data sheets and user's manuals and/or
- ❖ *Ap*BUILDER plus hypertext manuals
- ❖ Performance benchmarks
- ❖ *Model*BUILDER

mance to accomplish the job?" These are but a few of the questions an engineer ponders during the selection phase.

What if the engineer doesn't know what a peripheral is capable of doing? There are very few tools that help an engineer figure out what to do with a microprocessor or microcontroller peripherals, but they do exist. Evaluation board products are designed to "show off" micro-processor or microcontroller features. The trouble is that an engineer needs to understand something about it even before he can get started. It's kind of like looking up the word "physical" in the dictionary. Does it start with *ph* or *f*? How do you know it is *ph*?

There are also boards that help an engineer get started with common microcontroller functions. They too are evaluation boards, but have friendly user interfaces to simply point and click a mouse to drive microcontroller peripherals, giving the engineer an example signal for referencing. Because of the complexity of microprocessors and microcontrollers, engineers become intimidated to try new devices. In choosing devices for a new application, the tendency is to use a familiar device rather than to choose one for its cost effectiveness or technical merit. While choosing a device because of its familiarity may forego some of the learning process, in the long run, taking a little more time up front to select the "right" micro will save the engineer from trying to plug a square peg into a round hole.

Once an engineer has chosen a device or set of devices that "could" do the job, how does she find out whether the device has enough performance to achieve the task? This should be left up to the engineer and not the semiconductor companies. Sure, many semiconductor companies will be glad to take the device for a "test drive," but it's not their job on the line if something goes wrong.

Semiconductor companies need to reduce the learning curve. Engineers need to select the right device for the job and check its performance in the application, without spending precious time learning about how to manipulate a peripheral to perform a task.

For this reason, Intel created such tools as *Ap*BUILDER and *Model*BUILDER. These tools are designed to move an engineer down the learning curve quickly and with little pain.

*Model*BUILDER helps the engineer find the processor's true performance in the application without having to learn about the processor first. The engineer simply selects from a set of performance templates that represent functions that a typical embedded application would perform and tells it how often these routines need to be performed. It responds with the device's true performance in the application. This whole task is performed by a click of a mouse, without picking up a user's manual or handbook, and within minutes.

By using *Model*BUILDER, the engineer can make a quick decision whether this device has enough merit to continue. It is very frustrating to learn all about a device, just to find out that it can't do the job.

*Ap*BUILDER is designed to take the engineer beyond the benchmarking stage and into the discovery stage. *Ap*BUILDER breaks the microcontroller or microprocessor down into its basic elements. Hypertext manuals are available to observe the benefits of the device. When the engineer is ready to write code, *Ap*BUILDER allow automatic code generation from simple clicks of the mouse to tell the expert system what the engineer wishes to do with this peripheral.

From device understanding into code development, *Model*BUILDER and *Ap*BUILDER transform a novice into an expert in a short period of time. This translates to no lost time spent on inadequate devices, no lost opportunities because of misunderstanding, and no pain in learning new things!

Continued on page 14

The Stages Within the Selection Phase

Search	Use a line card or handbook to help narrow the search to a few devices. If the application is data control, look to the 80C186XL/EA/EB/EC or Intel386™ SX/CX/EX embedded microprocessors for solutions. If it is event control, look to the MCS® 48, MCS 51, or MCS 96.embedded microcontrollers.
Mini-benchmarking	Use *Model*Builder to get a "feel" for the device's performance within the application within minutes, without understanding the device.
Discovery	Use *Ap*BUILDER to start learning about the device and generate code for any peripheral.

Tools Required for System Evaluation

- Hardware and software manuals: *Ap*BUILDER and hypertext manuals
- Software: evaluation copies of software, assembler, compiler, or both, linker, and monitor/debugger
- Hardware: evaluation boards/kits, logic analyzer (optional), oscilloscope, DVMs, sockets, prototyping board (optional).

or:

- Project Builder kit

Continued from page 13

2. System Evaluation Phase: The mini-development phase.

System Evaluation Phase

The system evaluation phase is used to decide whether to use the device. Many times this phase is eliminated because of time constraints, but it often could save the company valuable resources.

While *Model*BUILDER gave a first-order approximation of device performance, this phase achieves a true assessment of CPU and peripheral performance in the engineer's application. To accomplish this task, the engineer needs to program the device in question with several routines that will indicate (benchmark) its performance. Several semiconductor manufacturers would rather they do this task. The common phrase from a semiconductor company is, "Send me your code and I will have my applications engineers code it for you and tell you how fast it runs." What happened to evaluating the tools in addition to the silicon?

During the system evaluation phase, engineers need to perform a mini-development cycle. They evaluate the device further and also evaluate the tools required to implement a design. Software tools such as assemblers, compilers, linkers, librarians, simulators, and debuggers are used. Even hardware tools like prototyping boards, evaluation boards, logic analyzers, and meters are used.

Many semiconductor companies put evaluation kits together to assist in the system evaluation phase. The trouble is that these kits are sometimes incomplete or not quite enough. Intel's Project Builder kits are incredible for accomplishing this system evaluation step.

The kit includes a hardware target board, retargetable symbolic debugger, an assembler and compiler capable of 8K of code development, *Model*BUILDER software, *Ap*BUILDER with the hypertext manual for the device, board schematics and schematic library, and timing analysis software with device libraries using Chronology's TimingViewer. A complete development kit for under $200 (for the 80C196KC/KD processors).

At the end of the system evaluation phase, the engineer has enough information to make a qualified decision whether to use this device. Without this phase, an engineer is taking a risk of encountering problems down the road.

It is highly recommended that this phase be performed when using a device for the first time, to avoid pitfalls during the full development phase. When using a familiar device, the engineer needs to investigate the tools available again. New items are introduced all the time and may make the development cycle quicker.

The Stages Within the System Evaluation Phase

Project Builder or Evaluation Kits	Evaluation and/or project kits allow the engineer to fully evaluate the product's capabilities and even learn a little more about the device.
Mini-development	The engineer writes several routines that assess the device's true performance in the application. CPU plus peripheral performance is a better indication than CPU performance alone.
Commitment	If the device meets the application requirements, the engineer makes a commitment to use it in the design.

3. Full Development Phase: Full steam ahead!

Formal Training

At this point, the engineer is in full development and has committed to using the device in the design. Because of this commitment, it is a good idea to acquire full knowledge of the device's features and functions. Many times this requires formal training.

In addition to formal training courses, semiconductor and tools manufacturers offer support hotlines, faxback systems, and BBS services to assist in the training and support process.

Consultants

Consultants can help in this phase as well as throughout the development cycle. These are individuals who have a working knowledge of the device. They can eliminate time in the initial development because of this prior knowledge. When using consultants make sure these things are understood:

1. Future application revisions may require re-employing these consultant unless proper transfer of consultant knowledge has taken place.
2. Make sure consultants fully document their work. Heavy emphasis should be placed in this area to avoid headaches down the road.
3. Allow for a little extra time at the end of the design in order for the consultant to transfer this information to the company employees.
4. Make sure the consultant is under a non-disclosure agreement if the design is secret.

Tools Required for Full Development Phase

- ❖ Complete software suite: assemblers, compilers, linkers, librarian, text editors, simulator (optional), real-time kernels or real-time operating system software, make utilities or ide, software library, fuzzy logic software, and performance analysis tools (optional).
- ❖ Complete product documentation: datasheet, user's manual, programmer's manual, errata sheets, ap-notes, and example code
- ❖ Hardware for debugging: emulator, logic analyzer (optional), oscilloscope, DVM, EPROM programmers (optional), and socket adapters
- ❖ Consultants (optional): board-level engineering services for board layout and fabrication (optional)

As part of the training process, it is a good idea to investigate any previously published material. Apnotes and technical tidbits are good sources for examples and can point the engineer in the right direction. Don't reinvent the wheel.

Software Development

Since a majority of the system's functionality is software driven, the software development stage is very important. Not to downplay hardware development, but more things can go wrong with the software than with hardware. The better the software development tools, the better the code developed.

Complete Software Suites

Software suites are becoming more and more popular. Software vendors are starting to bundle all of the software required for software development into one package. Don't forget text editors. Today's word processing software does a great job for electronic publishing but is not the right editor for code development. It's not that a knife won't work as a screwdriver but that the right tool for the job may make the task easier.

Integrated development environments (or IDEs) are the way of the future. PC code development has been enjoying this environment for years and the embedded world is starting to enjoy it. IDEs make code development more complete. They have built in text editors, assemblers, compilers, linkers, and can even cross over to a debugging environment that is linked with a software simulator or emulation hardware.

Assemblers, Compilers, and Linkers

An engineer can certainly develop all required code using a simple assembler, but with today's optimization technology and reasonably priced compilers, using a compiler in addition to that assembler might reduce the software development stage. Assemblers are great for routines that need to be really tight code or routines that are under strict time constraints.

Code development is done best when split into modules with the main routine calling these modules or modules calling modules. The point is that a linker resolves these inter-connections between modules. A software engineer may

Continued on page 16

Continued from page 15

have hundreds of these modules. Make sure the linker has the capability to handle a large number of object modules and libraries.

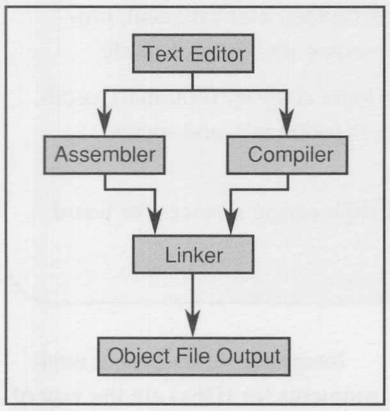


Figure 2.Code Development

Object File Format

The best assembler, compiler, or linker in the world is useless unless the debugger's loader can read the output file. Make sure the output format is compatible with the debugger used by the simulator/emulator. There are many interpretations of the standard output module format IEEE 695. Be sure to read the fine print on all the tools. Does the object file format contain symbolic information and references to data types? Later when debuggers are discussed, symbolic references to data and program make debugging code easy and straightforward.

Simulators

Simulators are nice for initial code development. Much of the code can be "wrung out" using simulators. There are many kinds of simulators, most of which simulate the device's CPU and CPU registers. While CPU simulators are good for CPU code development, when it comes to real-time debugging, a CPU simulator may not be as useful.

Some simulators also include simulations for interrupts, timers, I/O, and peripherals. These simulators are very useful for real-time debugging without silicon.

There are a couple other things to check about a simulator: Does it provide for synchronous and/or asynchronous "data trains" for input stimulus? Can the simulator results be directed to a file? This can be very useful when trying to track down a tough bug buried in the code.

Debuggers

While debuggers are software programs that run on computing platforms, they still need simulators, evaluation boards, emulators, or some type of hardware to make them useful.

A debugger is the interface between the device functionality (either simulated or actual) and the engineer. Debuggers many times are the identity of the hardware. For example, engineers may not see the hardware, but they certainly remember the keystrokes and screens used to manipulate the hardware.

Historically, emulator and simulator companies have written their own debugger interface programs. But with today's quick time to money, a standard debugger interface to any hardware simulator/emulator platform would reduce the learning curve. With this in mind, some emulator vendors are electing to use standard debuggers for their user interface. This way, engineers do not have to learn a new user interface for every device they use in designs.

There are a few tips to remember when checking out debuggers: Make sure the debugger has a built in assembler and disassembler. It will be very useful to perform simple code patches without recompiling code. Symbolic debugging is essential. Make sure the debugger can input the object file format with the symbolic information. Hex files do not contain symbolic information. While code development can be done without symbolic debugging, it makes the job a lot faster when in place. Make sure the debugger has the right capabilities, such as breakpoints, single-stepping, program and data tracing, and memory real-time display.

The more features the debugger has, the faster the code development will go (as a rule). The value of a good debugger is in the engineer's eye. A good one is worth a gold floppy disk.

Software Libraries

Software libraries can reduce debugging time. Many compiler and assembler vendors also sell software libraries. There are also vendors that sell libraries for specific functions, like serial protocol links and math routines. These routines have been fully debugged and tested.

User beware! Even though these libraries are debugged and functional, it is still the engineer's responsibility to make sure the code is correct.

Real-time Kernels and Real-time Operating Systems

Other useful software tools are real-time kernels and real-time operating systems (OSs). Having a basic operating system and simply applying these OS functions can greatly decrease software development.

Features like configurable kernels, multitasking, reentrant services, and real-time response times are important. Having access to the source library of the kernel or OS is also nice. Many real-time kernels understand the specific peripherals of the device and take advantage of its functional capabilities. Make sure the OS was designed specifically for the device you're designing with.

Fuzzy Logic Compilers

While this article will not fully explain the benefits of fuzzy logic usage in embedded designs, engineers around the world are finding the use of fuzzy logic technology and microcontrollers to be beneficial, especially in designs where not all variables are well known.

Fuzzy logic is a technology that enhances model-based system design using intuition and engineering heuristics. It represents the desired system behavior using elements of everyday language.

Fuzzy logic compilers work best on standard microcontrollers using all of the microcontroller features. They generate ANSI C or assembly language code specifically for a given architecture. In this way, they can take full advantage of the CPU instructions and peripherals.

Socket Adapters

During the hardware and software debugging stages, socket adapters are often used. Some socket adapters change one package type into another, while others allow for easy access to chip signals while it is plugged into a socket in the system.

EPROM programmers often use socket adapters to support different package offerings. The same signals are used to program a device. With different packages, those signals are just in a different place.

Continued on page 18

Definitions

(many terms derived from *The New IEEE Standard Dictionary of Electrical and Electronics Terms*, fifth edition, IEEE Std 100-1992)

*Ap*BUILDER

*Ap*BUILDER is a powerful design tool for the embedded control applications programmer. This product was created specifically to speed up the learning curve and reduce the total design time, no matter what level of processor experience. *Ap*BUILDER software provides peripheral design capabilities and direct manual referencing.

Block diagram screens give the engineer easy "point and click" programming of any device peripheral. Forget about digging it up in a paper manual, it is a mouse click away, directly into the section where it is referenced.

*Ap*BUILDER also generates the initialization code for all the device peripherals. This allows the engineer to get started on code development before understanding every bit in special function registers.

Assemblers

Assemblers are utility programs that translate symbolic assembly language instructions into machine instructions or data on a one-to-one basis. There are several types of assemblers: cross assemblers, absolute assemblers, relocating assemblers, and macro assemblers. Each provides the same type of one-to-one translation of assembly code into machine code.

Compilers

Compilers are utility programs that translate symbolic, high-level programming language instructions into machine instructions or data on a one-to-many basis. There are several types of compilers: cross compiler, root compiler, and incremental compiler. A cross compiler is a compiler that executes on one computer but generates machine code for a different computer. Most all microcontroller compilers are cross compilers. A root compiler generates a machine-independent, intermediate-level representation of the program. A root compiler must be combined with a code generator to be a full compiler for a particular device. An incremental compiler is one that translates each source statement as it is inputted or as the source statement is scanned. This typically refers to an on-line compiler or interactive compiler. Some debuggers have a built-in incremental compiler.

Debugger

A debugger is a computer program that is used to detect, locate, and correct faults in a computer program. It uses program breakpoints, data dumps, single-stepping, program tracing, data tracing, and memory modifications to perform this task. A debugger must also interface with the device running the program being debugged. This device can be in the form of a simulator, emulator, evaluation board, or simple device monitor, but it must be able to perform the aforementioned tasks to be effective.

Emulator

An emulator is a hardware system representation of a device. It accepts the same inputs and produces the same outputs as the device being represented.

Evaluation Board

A printed circuit board used to develop preliminary versions of hardware and/or software. Like prototyping, a evaluation board saves the design engineer time because it already has a basic functional system with program execution and debugging capabilities. Many evaluation boards have a separate hardware prototyping area for customized hardware development.

Evaluation Kits

Evaluation kits typically come with a hardware board for prototyping, a command line monitor or debugger, plenty of paper manuals for reference, example code, and sometimes an assembler. Evaluation kits are also designed for the system evaluation phase and can be very helpful in assessing the device's true performance in an application.

Hypertext Manuals

Hypertext technical documentation is available for all of the embedded 186, MCS® 96, MCS 51, and Intel386™ EX microprocessors. The complete user's manuals and data sheets have been electronically produced under the hypertext format in order to present the information in a clear and straightforward format.

Continued from page 17

Socket adapters can cause some trouble if they are not properly seated. Beware of the continuity problems that exist with adapters. Make sure they are secure before debugging starts.

EPROM Programmers

If the design contains an EPROM, most likely an EPROM programmer is required to burn the program into the EPROM.

EPROM programmers take hex or object files as input and use high-voltage signals to permanently burn the program into the memory.

Make sure the programmer used can read the hex or object file that the assembler, compiler, or linker outputs (i.e., Intelec Hex, Binary, etc.)

A useful capability of programmers is address offset before programming. This is important if the EPROM addresses don't line up with the system addresses.

EPROM programmers typically are used for several different devices (EPROMs, Flash memory, PLDs, microcontrollers) Make sure that the programmer supports the programmable device used in the design as well as in the package types. Socket adapters are OK for pin translations, but make sure the device is seated well before programming begins.

Emulators

While this article will not do them justice, emulators are by far the most useful tools an engineer will use. The more complex the device being emulated, the more complex the emulator (as a rule).

An emulator is used to replace the device under emulation (DUE) in a system. It, along with the debugger, make up the hardware and software debugging environment. The more features an emulator has, the faster the debugging process will go (provided the engineer is knowledgeable on these features).

There are some rules of thumb to follow with emulators:

1) Make sure the emulator has enough features to get the job done. (Features like program and data breakpoints, watch windows, disassembler, in-line assembler, single stepping, super stepping over routines, and program execution trace as well as data and I/O tracing.)
2) Many emulators have logic analyzer capabilities. While these features make it easier to debug using one tool, the lab may already have a logic analyzer. Buying these features in an emulator may waste a perfectly good tool in the corner.
3) Buy the maintenance and service contract. Like any piece of equipment, occasionally the emulator will go down. Murphy's Law states it will go down at the most critical time in the development cycle. To minimize the down time, spend the money.
4) Upgrades to emulators can be important. Many microcontrollers and microprocessors are simple peripheral or memory scalars of each other. Some emulators are designed to easily handle these proliferations at minimal cost.

Renting versus Purchasing

Emulators and logic analyzers can be expensive. For those times when buying is just not within the budget, many vendors offer rental programs. Rental firms also exist that offer a variety of emulators, meters, power supplies, oscilloscopes, and logic analyzers to choose from. The design gets the proper tool attention, and after the job is done, the tool doesn't sit in a corner collecting dust.

The Stages Within Full Development Phase

Formal Training	Using computer-based training, formal classroom training, or consultants. Not only device training, but tools training is important for efficient use of engineering time during development.
Software Development	Using compilers, assemblers, linkers, simulators, debuggers, software libraries, real-time OSs or kernels, fuzzy logic compilers, consultants, EPROM programmers, and emulators.
Hardware Development	Using prototyping hardware with logic analyzers, oscilloscopes, DVMs, emulators, socket adapters and consultants.
Design Layout	Using schematic capture software, PC board layout software, engineering services, and consultants.

Hardware Development

Hardware development requires tools for board design and layout as well as tools for DC parametric and AC timing analysis.

Schematic capture packages are used to draw circuit diagrams using standard device libraries. The outputs of these packages feed into board layout software as well as circuit analysis packages.

Although most engineers do timing analysis to make sure the design is stable, it is not a loved art. Having the ability to graphically manipulate and analyze timing information of a system would be helpful. Software packages are available today that make this task easier and less painful.

Board-level Engineering Services

After the engineer has properly checked out the design and made sure the pins are all connected to the right places, schematic capture software generates a netlist. This netlist is used to generate a PC board layout. PC board houses often have the capabilities to assist an engineer in the board layout and critical parts placement. They can offer layout tips about power planes, oscillator circuits, and RFI reduction.

Consultants

Consultants are also used in the hardware development stage. They usually have a working relationship with board houses and often have schematic capture and layout software. Consultants have complete knowledge of a device and can offer suggestions that could save valuable time during hardware debugging.

Continued on page 20

Definitions (Continued)

Libraries and Librarian

A librarian, like a linker, is a utility program that translates assembled or compiled modules into object modules. It translates object modules into a library of commonly used modules that are often used in code development. Many software vendors supply common libraries for floating point arithmetic, trigonometric functions, and common peripheral functions.

Linker

A linker is an optional translation utility program for taking assembled or compiled modules and linking them together. It resolves any cross references between the modules and generates a single machine language object module. Its input is one or many assembled/compiled modules, while its output is one absolute module.

Logic Analyzer

Logic analyzers are complex instruments capable of performing traditional logic-level, timing, and glitch-detection tasks. Many logic analyzers also provide sophisticated disassembly for microprocessors and microcontrollers. Some do high-speed asynchronous timing analysis as simple oscilloscope functions. Logic-level memory buffers are used to analyze program execution and data flow. The larger the logic memory buffer, the better. It is important to use an analyzer that has a complex logic triggering mechanism with loop counters, qualifiers, and logical AND and OR functions. These function are always useful in the program debug stage.

Make Utility

A make utility is a computer program that aids in the assembly, compilation, and linkage of software modules. Not to be confused with IDE, make utilities only make the assembly, compilation, and linkage process simpler. They are not intended to be IDEs.

***Model*BUILDER**

For the first time, a tool was created for the selection phase of the design cycle. Not just a line card, data sheet, and a user's manual. A completely integrated performance tool that helps an engineer decide whether a device is right for the application, based on the performance of the chip. Today *Model*BUILDER works for the MCS® 96 microcontrollers. It allows an engineer to select from a set of "real" performance templates. These templates closely resemble functions that can be performed using the MCS 96 microcontroller, such as A/D scanning, I2R filtering, tachometer, 3 phase motor control, and matrix manipulations, just to name a few.

These templates can be compiled for total system integration. *Model*BUILDER is intended to give a 1st order approximation of the MCS 96 microcontroller's performance within any application within minutes.

Integrated Development Environment (IDE)

An IDE is a computer program that assists in software development by bringing all of the software tools such as text editor, assemblers, compilers, linker, debugger, and simulator together within one environment.

Project Builder

This is Intel's kit specifically designed to assist in the system evaluation phase of development. It contains all the tools required to perform a mini-development cycle, including teaching aids like *Ap*BUILDER and *Model*BUILDER; software suites like assembler, compiler, linker, make utility and symbolic debugger; hardware target system for prototyping; and hardware analysis tools like full schematics and timing analysis.

Prototyping Boards

A printed circuit or wire wrapped board used to develop a preliminary version of part or all of the hardware. It is developed to permit feedback, determine feasibility, or investigate timing or other issues related to a design.

Real-time Operating Systems

Operating systems are a collection of software, firmware, and hardware elements that control the execution of a computer program. They provide such services as resource allocation, input/output control, data management, and job control in a real-time manner. Real-time means that computations are performed during the actual times that external asynchronous events occur in order to control, monitor, or respond in a timely manner to the external process.

Simulator

A simulator is a computer program that is used to represent the function of a system. Many microprocessor and microcontroller simulators only simulate the CPU functionality. A good simulator will also simulate interrupts and event peripherals.

Continued from page 19

4. Production Phase: Shipping units; still need tools.

Tools Required for Production Phase

- ❖ Gang EPROM Programmers
- ❖ Engineering Services: PC board manufacturing and test houses.

Production Phase

Even when the product is in production, tools are required. Tools to analyze field failures, aid in the production process, and even make small production line "tweaks" prove handy.

Gang EPROM Programmers

If the system has an EPROM on board or the microcontroller needs to be programmed before or even during production, an EPROM programmer is needed. Gang EPROM programmers are used to program more than one device at a given time. Sometime an engineer requires that the device be programmed during production. This too may require special programming hardware. Many microcontrollers can be programmed during production. Check the device specifications and programming modes.

ONCE Mode and Emulators

Analyzing field failures can be performed using many of the same tools used in development. Logic Analyzers, oscilloscopes, DVMs, and emulators are particularly useful. Some devices have built-in capabilities to assist in the failure analysis. Most microcontrollers and microprocessors built by Intel have a mode called ONCE, which stands for ON-Circuit Emulation. A special pin of the device is driven to a particular state just before and immediately after an active RESET pulse. When this condition exists, the device's outputs are totally disabled. This allows other hardware, like emulators, to drive lines that would normally be driven by the microprocessor.

Many emulators support the ONCE mode. They use a special adapter to clamp onto the device soldered on the PC board. They put the device on the board into ONCE mode and drive the system using the emulator. Many manufacturers use the ONCE mode and emulators for board testing. Check into the emulator's capabilities for ONCE mode support.

Production Services

If the design requires manufacturing and testing of PC boards and this not a known capability of the designer, many companies supply production services. These companies fabricate PC boards, procure parts, stuff the boards, wave solder, and test the boards for defects. Some even box and ship directly to customers.

Whichever production house is used, make sure proper handling, manufacturing, programming, and testing specifications are in place before starting production. Many production houses will assist in debugging manufacturing failures and help streamline designs to reduce production time.

Intel's Commitment to the Development Cycle

Whether in the selection, system evaluation, full development, or production phase, engineers find that tools are essential in completing the development cycle. Intel is committed to making the development cycle one of the easiest in the industry.

Intel is working hard to help our third party tools vendors. We are making our bondout technology available, assisting them in the technical development of tools, and driving standards in the tools industry. We feel these things will create accurate, quality, affordable, and technically superior tools for all of Intel's valuable customers.

The *Development Tools* handbook (order number 272326) is part of that commitment. It presents a list of third party vendor tools that support Intel's embedded microcontrollers and microprocessors. For over 25 years, Intel has been committed to its customers. We plan to carry on that tradition. ■

New Embedded Control FaxBack* Service Documents

New/Updated as of June, 1994

Title	Number
Product Line Card: 80960 Microprocessor Family (includes table of features for entire family)	2033
80C196KB/83C196KB: B-0 to C-1 Stepping Conversion	2316
MCS 96: Understanding SSIO Clock and Data Transmissions	2317
Flash Memory Technical Support Summary (All inclusive summary)	2204
InfoGuide: Embedded Intel386 Microprocessor Family	2720
Press Release: Introduction of the iFX730/iFX760	2321
News Release: 27% Price Reduction on iPLD22V10-7	2322
Flash Memory Card Programming Support	2200
28F256A: Programming Support	2575
28F512: Programming Support	2576
28F010: Programming Support	2577
28F020: Programming Support	2578
28F001BX: Programming Support	2579
28F200BX, 28F002BX: Programming Support	2580
28F400BX, 28F004BX: Programming Support	2581
28F008SA: Programming Support	2582
28F032SA: Programming Support	2222
Prototype: 44-ld SOP Dimensions/Land-Pad Layouts: Prototype Socket and Device	2228
80C186: Development Tools Line Card (includes Product line Card)	2624
iFX760: Data Sheet (10ns FLEXlogic FPGA with SRAM option)	2319
iFX730: Data Sheet (10ns FLEXlogic FPGA Family with SRAM option)	2320
Product Line Card: Embedded Intel386 Microprocessor Family	2764
Data Sheet: Intel386TM CX Embedded Microprocessor	2753
Data Sheet: Static Intel386TM SX Embedded Microprocessor	2754
Press Release: SystemSoft® to Enhance and Distribute Microsoft® FFS	2231
Inductors: Micro Thin SMT Inductors for Type I and II PCMCIA Cards	2232
Converter: Micropower Step-Up DC/DC Converter for TYPE I PCMCIA Cards	2233

Title	Number
News Release: New Intel 16- and 32-Mbit Flash Memories Offer Highest Density Rewritable Storage for Mobile Computing	2234
28F016SA: Programming Support	2213
MCS 96: Digital Filter Techniques Using the 80C196	2318
InfoGuide: Embedded Intel386TM Common Questions and Answers	2766
Embedded i386: Hardware and Software Development Tools Line Card	2765
i386EX: Evaluation Board Fact Sheet	2767
i386EX: Evaluation Board Schematics	2768
Fact Sheet: Overview of the PC/104 Standard, and vendors	2769
i386EX: Compatibility with PC/AT Architecture	2770
i386CX: Chipset Possibilities for the Intel386CX	2771
i386EX: Systems and Software Intel386EX™ Simulation/Debugger	2772

Customer Education

The training schedule for the second and third quarter of 1994 includes the following embedded courses:

- 8051 Microcontroller Family (ED3030)
- MCS 96 BH/KB/KC/KD Microcontroller Family (ED3040)
- Introducing the Intel386™ EX Microprocessor Architecture (ED3022)
- Applying the Intel386™ EX Microprocessor Architecture (ED3021)
- Introduction to Fuzzy Logic Technology (ED3046)
- Fuzzy Logic Design (ED3047)
- iRMX® Operating Nucleus Concepts (ED3070)
- iRMX® Windows™ Operating System (ED3075)
- i960® KA/KB/CA Embedded Processors (ED3050)
- i960® Superscalar CA/CF Microprocessors (ED3051)

and one **NEW 1994 COURSE:**

- 80960JX Series (ED3053)

Most courses are taught in Phoenix, Arizona. To register or to get more information, please call Customer Training at 1-800-234-8806.

GLAD YOU ASKED

Q's and A's

Article ID# 0908

8XC196 Q&A's

Q: Are the 8XC196NT and 8XC196KD pinouts the same?

A: No, they are not pin-to-pin compatible. Although both are available in 68-lead PLCC packages, their signals are different.

Q: Can register RAM be used for code?

A: No, code cannot execute from register RAM.

Q: Is external logic needed for 8XC196NT memory accesses to page 00 SFRs?

A: The register file and SFRs in page 00 are directed to on-chip memory, so external logic is not necessary.

Q: For the 8XC196NT, how can I access locations 0FF0000H through 0FFFFFFH when I am using external memory instead of internal memory (EA# is low on reset)? There are only enough address lines to access up to 0FFFFFH.

A: Addresses directed to external memory on page 0FFH will be seen externally as accesses to page 0FH. Internally, the processor will see these as different locations. Consequently, you must be very careful not to overlap different code or data to these locations.

Q: When using the 8XC196NT, how can you use the PTS for transfers when the source and destination have only 16 bits?

A: The PTS can move data from one page to another only if the register file is either the source or the destination. Since the register file is effectively mapped into all pages when nonextended instructions are used, data can be moved between the register file (page 00) and the page pointed to by EP_REG.

Q: I have recently converted from the 80C196KB B-0 stepping (change indicator B) to the C-1 stepping (change indicator F or G) and have noticed differences. Were there any changes in this process change?

A: Yes, there were some AC/DC and A/D specification changes in the process change for the CPU/ROM 8XC196KB. For a list of these differences, see FaxBack document #2316.

Intel386™ EX Embedded Microprocessor Q&A's

Q. What peripherals are built into the Intel386™ EX Microprocessor?

A. The microprocessor contains the following integrated peripherals:

- clock and power management unit
- integrated chip-select unit with eight chip-select lines
- interrupt control unit containing two 82C59A modules connected in cascade
- timer/counter unit with the same functionality as the 82C54 counter/timer, providing three timer/counters
- watchdog timer unit
- asynchronous serial I/O unit that is equivalent to the National Semiconductor NS16450 and INS8250 and contains two full-duplex asynchronous serial channels
- synchronous serial I/O unit for simultaneous bidirectional communications
- parallel I/O unit containing three 8-bit, general-purpose bidirectional I/O ports
- DMA controller containing a two-channel DMA that can be used as an 8237A-like controller
- refresh control unit that greatly simplifies DRAM refresh
- JTAG test-logic unit that is fully compliant with IEEE Standard 1149.1

For detailed information, consult the *Intel386™ EX Embedded Microprocessor* data sheet (order number 272420) and the *Intel386™ EX Embedded Microprocessor Hardware Reference* (order number 272485).

Q. What are the differences between the Intel386™ SL microprocessor and the Intel386 EX embedded microprocessor?

A. The Intel386 SL microprocessor and the Intel386 EX microprocessor have the same static internal core and both have integrated system management mode, chip-select unit, and integrated DRAM refresh controller. The Intel386 EX microprocessor adds several peripherals that are not available on the Intel386 SL microprocessor (see question above). The Intel386 EX processor does not have the integrated cache controller and tag RAM and it does not offer the direct ISA bus interface that the Intel386 SL controller did. However, it will be possible with only slight hardware and software modifications to recreate the ISA bus. (This hardware may soon be provided in the form of an FX740 or a companion chip.) The Intel386 EX processor is packaged in a 132-lead PQFP, whereas the Intel386 SL processor is packaged in a 196-lead PQFP. Although it is not a pin-for-pin replacement, the Intel386 EX processor is code-compatible with the Intel386 SL processor and should be considered the replacement of choice.

Q. What is PC/104?

A. PC/104 is a new standard board based on the PC and PC/AT architecture. It is simply a smaller version (3.8" x 3.0") of the standard PC bus board. Its purpose is to allow PC-compatible architecture to be compact enough to be embedded and still be able to use standard components. Intel has announced cooperative efforts to work with Ampro, a leading producer of PC/104 hardware, to develop the embedded market for PCs. All information on what is planned for embedded Intel386 processors to PC/104-based systems must come from the PC/104 vendors. For more information about PC/104 standards and a complete list of PC/104 vendors, order FaxBack document #2769 or contact the PC/104 Consortium at 415-903-8304.

Q. Can the Intel386™ EX, CX, and SX microprocessors be configured to be completely PC/AT compatible?

A. Yes. See FaxBack document #2770 or Chapter 4 and Appendix B of the *Intel386™ EX Embedded Microprocessor Hardware Reference* (order #272485).

Q. PC/AT architecture uses two 8237A DMA controllers, connected in cascade, for a total of seven channels, while the Intel386™ EX microprocessor has only two channels. What effect does this have on DOS compatibility?

A. The Intel386 EX microprocessor is designed so that it can easily be configured for full DOS compatibility. All versions of DOS 3.2 and higher use only one DMA channel, for the floppy disk controller. Thus, the two channels supplied by the Intel386 EX processor are sufficient. If older versions of DOS are used or if more than two channels are required, two 8237A DMA controllers can be easily connected to the Intel386 EX processor and can replace the on-board, two-channel DMA controller via an on-board relocation register, yielding yielding the PC/AT standard of seven DMA channels. For details, see Chapter 4 and Appendix B of the *Intel386™ EX Embedded Microprocessor Hardware Reference* (order #272485) or FaxBack documents #2755 and #2770 (document #2770 contains the same information as Appendix B of the hardware reference). Also refer to AP-499, *Introducing Intel's Family of Embedded Intel386™ Microprocessors*, (order #272425) for further information about DOS compatibility. ■

ERRATA AND CHANGE IDENTIFIERS

Article ID #0909

Change identifiers have been used since 1990 to distinguish revisions, or steppings, of embedded control devices. Older devices have no change identifiers. On most devices, the change identifier is the last character in the FPO number, which is typically a nine-character code on the second line on the top of the device. An example FPO number is "L1234567D," in which "D" is the change identifier. On some devices, such as the 8XC51SL-BG, the change identifier is a separate line item and uses several characters. For example, change identifier "SW011" identifies the B-3 stepping of the 8XC51SL-BG.

This article lists change identifiers, errata, and design considerations for recent steppings of embedded control products. For many of these devices, complete errata listings or explanations are available from the FaxBack* service. The "Ref" column in each table lists FaxBack service document numbers for errata lists and explanations, and "For More Information" at the end of this article has a complete list of related document numbers and titles.

MCS® 51 Microcontroller Family Errata and Design Considerations

This list covers the most recent steppings of the MCS® 51 microcontrollers. A complete list of the errata for all steppings is available on the FaxBack service (document #2632).

Table 1. MCS® 51 Microcontroller Family Errata and Design Considerations

Device	Step	Change Identifier	Errata and Design Considerations	Ref.
8051AH/8031AH	C	A	1. External interrupt 0 errata	2154 2161
	C-3	B	No known errata	
80C51BH/80C31BH	C	none	1. Reset lockup problem 2. High IPD if C does not equal B.7 before going into powerdown 3. ROM verify mode fails 4. Steam passivation problem on plastic parts	
	C-1	none	1. High IPD if C does not equal B.7 before going into powerdown 2. ROM verify mode fails	
	D	D or 2	No known errata	
87C51	D	A	No known errata	2106
80C52/80C32	C	none	1. RST/ONCE mode problem	
	A	A	No known errata	
83C51FA/80C51FA	C	none	1. PCA errata	2528
87C51FA	C	none	1. RST/ONCE mode problem 2. PCA errata	2528
	D	A	No known errata	2107
8XC51FB	A	none	1. PCA errata	2528
	B	A	No known errata	2111

Table 1. MCS® 51 Microcontroller Family Errata and Design Considerations (Continued)

Device	Step	Change Identifier	Errata and Design Considerations	Ref.
8XC51FC	A	none	1. Port 1, 2, 3 problem — asychronous port reset not supported 2. Failed ESD qual testing	2028
	B	none	No known errata	
8XC51GB	B	none	1. Reset polarity changed to active low 2. Port 1 reset value changed to all zeros	2032 2032
	B-2	none	No known errata	
8XC152JX	B	none	1. AE/RDN race condition 2. Receive FIFO is not cleared when receiver is enabled 3. DMA errata 4. SDLC flag recognition errata	 2030 2118 2035
	C	none	No known errata	
8XC51SL-BG	B-3	SW011	1. GATEA20, RCL hardware speedup processing 2. Powerdown current stabilization 3. Port 2 address mux 4. KSI powerdown wakeup interrupt 5. Reset errata	2008 2114
	B-4	SW062	1. GATEA20, RCL hardware speedup processing 2. Powerdown current stabilization 3. Port 2 address mux 4. KSI powerdown wakeup interrupt	2008
8XC51SL-AH/AL	A-0 A-1 A-2	AA BA CA	1. Power-down current stabilization 2. System power management errata	2048

MCS® 96 Microcontroller Family Errata and Design Considerations

This list covers the most recent steppings of the MCS® 96 microcontroller. Complete lists of the errata for all steppings are available on the FaxBack service for the 8X9XBH (#2134), the 8XC196KB (#2548), the 8XC196KC (#2136), the 8X196KD (#2315), the 8XC196KR (#2527), and the 8XC196NT (#2178).

Table 2. MCS® 96 Microcontroller Family Errata and Design Considerations

Device	Step	Change Identifier	Errata and Design Considerations	Ref.
8X9XBH	D	D	1. Indexed 3-operand multiply 2. HSI FIFO 3. Reserved location 2019H 4. RESET and the QBD pins 5. Software RESET timing 6. Using T2CLK for Timer2	2134
	E	E	1. Indexed 3-operand multiply 2. HSI resolution 3. Reserved location 2019H 4. Reserved location 201CH	 2631 2140

Table 2. MCS® 96 Microcontroller Family Errata and Design Considerations (Continued)

Device	Step	Change Identifier	Errata and Design Considerations	Ref.
8XC196KB 8XC196KB10/KB12	All		1. HSI 8/9 State 2. CMPL with R0	2548
	B	B	1. Divide during HOLD or READY 2. SIO framing error 3. SIO RI flag 4. DJNZW instruction 5. ALE glitch 6. HSI_MODE divide-by-eight	 2568 2192
8XC196KB/KB16	B	B, D	1. Divide during HOLD or READY 4. Missed EXTINT P0.7 3. HSI_MODE divide-by-eight 4. Oscillator sensitivity	2122 2049 2192
	C-0 C-1	E F, G	1. Missed EXTINT P0.7 2. HSI_MODE divide-by-eight	2049 2192
8XC196KC			The number following each entry in this list is a cross-reference to the applicable section of FaxBack service document #2136.	2136
	all		Design Considerations 1. Indirect shift count value 116 2. Write cycle during Reset 147	
	B-1	B	1. Divide error during hold 109 2. NMI during PTS skips address 123 3. QBD glitch during powerup 163 4. ONCE mode entry 214 5. Oscillator startup 215 6. Reset hysteresis 216 7. Missed EXTINT P0.7 8. HSI_MODE divide-by-eight	 2049 2192
	B-3 B-3	D or E D or E	1. Divide error during hold 109 2. NMI during PTS skips address 123 3. QBD glitch during powerup 163 4. Reset hysteresis 216 5. Missed EXTINT P0.7 6. HSI_MODE divide-by-eight	 2049 2192
	D	H,J,L,M	1. Missed EXTINT P0.7 (80C196KC only) 2. HSI_MODE divide-by-eight. 3. IPD Hump	2049 2192 2311
8XC196KD	A-1	B	1. Missed EXTINT P0.7 2. HSI_MODE divide-by-eight 3. IPD Hump	2049 2192 2311
	B	D,E	1. Missed EXTINT P0.7 (83C196KD only) 2. HSI_MODE divide-by-eight 3. IPD Hump	2049 2192 2311
8XC196KR/JR/KQ/JQ	A, C	A, C	Design Considerations 1. P6_REG not updated immediately	2527

Table 2. MCS® 96 Microcontroller Family Errata and Design Considerations (Continued)

Device	Step	Change Identifier	Errata and Design Considerations	Ref.
8XC196KR/JR/KQ/JQ (Continued)			2. Write cycle during reset 3. EPA timer reset/write conflict 4. Valid time matches 5. CLKOUT 6. Indirect shift operation 7. Internal RAM powerdown leakage 8. A/D latchup 9. INST operation 10. KQ/JQ memory map	
	A	A	**Errata** 1. Oscillator noise sensitivity 2. Slave programming mode 3. A/D abort 4. PTS with other interrupts 5. PTS/NMI conflict 6. Data output register cleared when mode register is written 7. Divide error during Hold/Ready 8. SIO Mode 0 9. Remap mode on EPA3 10. Serial port framing error 11. EPAIPV value multiplied by 2 12. EPA_MASK1/EPA_PEND1 must be written as words 13. Interruptable block move (BMOVI)	
	A, C	A, C	1. Ioh2 = – 6 μA	
	C,D	C,D	1. Cannot access external locations 1B00H-1BDFH	
8XC196NT	D	D	Design Considerations In bus controller modes 1 and 2, in 8-bit bus mode, the upper address lines need to be latched	
8XC196MC/MD	B	B	No known errata	
8XC196NP	A	A	1. Illegal Opcode interrupt vector not taken.	

8XC186/8XC188 Family Errata and Design Considerations

Table 3. 8XC186/8XC188 Family Errata and Design Considerations

Device	Step	Change Identifier	Errata and Design Considerations	Ref.
80C186A, B	none		1. Non-contiguous Interrupt Acknowledge cycles 2. ERROR# processing during FWAIT instructions 3. Input high voltage requirement on SRDY and ARDY pins 4. Interrupt Status Register (DHLT and Timer Interrupts) 5. Bus preemption errata (HOLD/HLDA protocol) 6. 80C188 RFSH# pin output timing	2096
80C186XL	A	none	Never put into production	
	B	A	1. INTx/INTAx in Cascade Mode	2025
	C	B	No known errata	
80C186EA/80L186EA	A	A	1. Low hysteresis on RESIN# pin 2. TEST/BUSY#, RD#/QSMD#, LCS#, and UCS# input low voltage 3. INTx/INTAx in Cascade Mode	2025
	B	B	1. INTx/INTAx in Cascade Mode	2025
80C186EB/80L186EB	A	A or none	1. Entry into ONCE mode 2. Low hysteresis on RESIN# pin 3. SINT1 input not latched internally 4. Ready input during INTA# bus cycle 5. CLKOUT transitions on the rising of CLKIN instead of the falling edge 6. I/O ports 1 and 2 initialize to Port instead of Peripheral function (documentation error) 7. INTx/INTAx in Cascade mode	2025
	B	B	1. INTx/INTAx in Cascade Mode	2025
80C186EC	A	A	1. Early exit from Reset (with high Vcc, or low temperature, or both) 2. Powersave Mode initialization at Reset	
	B	B	No known errata	

For More Information

The following FaxBack* service documents contain errata lists and explanations.

Title	Number
MCS® 51 Controller Errata	
MCS 51: Errata and Design Considerations	2632
MCS 51: 1991 8 bit Microcontroller Book Errata	2123
MCS 51: 1992 8-bit Datasheet Errata	2165
MCS 51: FX-Core ALE Disable Datasheet Errata	2308
8XC51FX: QFP Pin 39 Change	2124
8XC51FX: PCA / TIMER2 Errata	2528
8XC51FX: Hardware Description Errata	2017
87C51FA: D Step	2107
87C51FB/83C51FB: B Step	2111
87C51FC: Data Sheet Errata I	2024
87C51FC: Data Sheet Errata II	2020
87C51: D Step	2106
87C54/80C54: Data Sheet Errata I	2022
87C54/80C54: Data Sheet Errata II	2021
8051/31AH: Shrink C-Step External Interrupt 0 Errata	2161
80C31/51BH: D-Step Marking	2015
87C51GB: A-1 & B Errata & Design Consider	2032
8XC51SLAH/AL Errata	2048
80C51SL-BG: Errata Version 2.6	2008
80C51SL-BG: Product Preview Data Sheet Errata	2630
80C51SL-BG: Errata	2130
80C51SL-BG: Reset Errata	2144
80C152: SDLC Flag Recognition Bug	2035
8XC152: External Demand DMA Bug	2129
8XC152: Global Serial Channel Bug	2030
8XC152: DMA Bug	2118
8XC152: Errata and Clarifications III	2043
MCS® 96 Controller Errata	
8XC196KB/KC/KD,8XC198,8XC194: HSI Events (9 or greater)	2052
Project Builder: Manual Errata	2170
8XC196KB: History and Errata	2548
* 80C196KB/83C196KB: B-0 to C-1 Stepping Conversion	2316
8XC196KB: ALE Glitch	2568
8XC196KB/KC/KD: HSI_MODE divide-by-eight	2192
8XC196KB/KC/KD: Missed EXTINT Interrupt Problems on PO.7	2049
8XC196KC: 1991 Handbook Errata	2131
8XC196KC: HSI PTS Handbook Errata	2141
MCS® 96 Controller Errata (Continued)	
8XC196KC: Bug List	2136
8XC196KC/KD: Powerdown Current (IPD) Hump	2311
8XC196KC/KD: 1992 User's Manual Errata	2570
8XC196KD: Bug List	2315
8XC196KR/JR/KQ/JQ: Errata	2527
8XC196KR Eval Board EPA Remap Warning	2120
8XC196: Hold/Ready DIV/DIVB Problem	2122
8XC198 and 8XC196KB/KB16: Data sheet Errata	2569
8X9XBH, 8X9XJF: Bug List	2134
8X9X: Reading 201CH Bug	2140
186 Controller Errata	
80C186: C186/188 Compatibility with 80C186XL / C188XL C-Step	2132
80C186: AMD 80C18x to INTEL 80C186/186XL Comparison	2540
80C18x XL/EA/EB: INTx/INTAx# Errata	2025
EV80C186EB: Eval Board Manual Errata	2158
EV80C186: 186 Family Eval Board Errata	2592
186XL/EA/EB/EC: 186 Family User's Manual Errata	2603
80C186/C188: B Step Technical Bulletin	2100
80C186/C188XL: B-Step Technical Bulletin	2101
80C186/C188EA: A-Step Technical Bulletin	2102
80C186/C188EB: A-Step Technical Bulletin	2103
80C186/C188EB: B-Step Technical Bulletin	2104
80C186/C188EC: A-Step Technical Bulletin	2105
80C18xEC/80L18xEC: B-Step Technical Bulletin	2057
Embedded Intel386™ Microprocessor Errata	
Errata: Intel386 EX A-Step Errata	2762
Document Corrections: Intel386EX Corrections/Additions to EAS Rev. 3.0	2193